

Queuing Theoretic Analysis of Power-Performance Tradeoff in Power-Efficient Computing

Yanpei Liu, Stark C. Draper, Nam Sung Kim

Electrical and Computer Engineering, University of Wisconsin Madison

Email: {yliu73@, sdraper@ece, nskim3@}wisc.edu

Abstract—In this paper we study the power-performance relationship of power-efficient computing from a queuing theoretic perspective. We investigate the interplay of several system operations including processing speed, system on/off decisions, and server farm size. We identify that there are oftentimes “sweet spots” in power-efficient operations: there exist optimal combinations of processing speed and system settings that maximize power efficiency. For the single server case, a widely deployed threshold mechanism is studied. We show that there exist optimal processing speed and threshold value pairs that minimize the power consumption. This holds for the threshold mechanism with job batching. For the multi-server case, it is shown that there exist best processing speed and server farm size combinations.

Index Terms—Power-efficient computing, queuing theory, data center network

I. INTRODUCTION

Large-scale data center networks have gained tremendous usage nowadays. Applications running inside such clustered servers include web searching, e-commerce, and compute-intensive applications. However, today’s data centers spend a large amount of capital on power usage and other associated infrastructures. Around 40% of total operation cost is related to power distribution, cooling and electricity bills [1]. In 2005, the total data center power consumption was 1% of the total U.S. power consumption and caused emissions as much as a mid-sized country such as Argentina [2]. Emphasizing the importance of these issues, we note that recently the U.S. Environment Protection Agency raised concerns to the Congress about the growing power consumption in data centers [3].

Much power consumed by data centers is wasted: servers on average are only 10–50% utilized [1], [4]–[6]. Low utilization is epidemic to data center operations due to strict service level agreements on peak workload provisioning. However, due to the lack of “power proportionality”, an idling server still consumes 60% of its peak power, drawn mainly in peripherals such as DRAM, hard disk drivers (HDDs), network interface card (NIC), etc. Thus, to conserve power it is preferable to shut down servers. When considering server farms consisting of multiple servers, jobs can be consolidated into a few servers so that the rest can be shut down. Server on/off decisions are often made in conjunction with processing speed adjustments. Dynamic voltage and frequency scaling (DVFS) is a conventional processing speed adjustment technique that changes the processor’s clock frequency (and thus the speed of computation) according to workload conditions in order

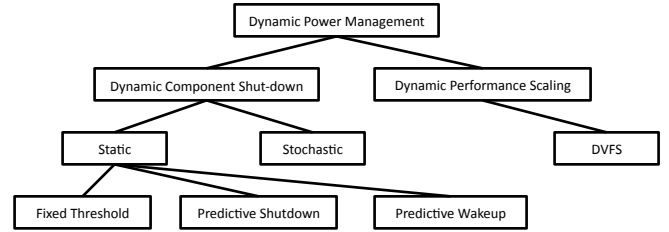


Fig. 1. Dynamic power management category [7]

to reduce power consumption. Server on/off decisions (also known as dynamic component shut-down) and processor speed adjustment (also known as dynamic performance scaling) can be categorized in Figure 1 (see Figure 5 in [7] for a complete diagram).

Our results tie into many earlier works, both in the computer architecture and queuing theoretic communities. The authors in [5] study a power saving method that shuts down servers when they are in idle and characterize the power-delay tradeoff from a queuing theoretic perspective. However they do not consider the performance scaling in their theoretic analysis. The authors in [8] investigate power reduction possibilities for jobs that demand fast response. They suggest that system-wide coordinated power management provides a far better power-latency tradeoff than individual uncoordinated decisions. The work in [9] also make similar statement. The authors study power management for MapReduce tasks, suggesting that all nodes in a MapReduce cluster should be powered up and down together rather than individually in a distributed fashion. The authors in [10] highlight the challenges of avoiding negative power saving. Negative power savings occur when the overhead of implementing the power-savings mechanism exceeds the resulting savings, thus costing the system extra power. They suggest guard mechanisms to monitor negative power savings and performance degradation caused by those power saving routines. The impact of data center size on power efficiency is evaluated in [11]. Most of the above works consider variants of a fixed threshold mechanism. In such mechanisms a server is shut down whenever it exceeds some idleness threshold. Stochastic on/off decisions are studied in [12] and stochastic optimization methods are also used in [13], [14]. For other related works on predictive shut-down and wake-up, see [7].

Surprisingly, although component shut-down and perfor-

mance scaling are widely used mechanisms in power-efficient computing, little is known from the queuing theoretic perspective, especially when component shut-down is jointly considered with performance scaling. The power-performance tradeoff in these settings is not well understood. This often results in suboptimal designs. We aim to study the fundamental interplay between these system operations including processing speed, on/off decisions, and server farm size from a queuing theoretic point of view. Our results yield clear design guidance. One result demonstrates that there are sweet spots in power-efficient computing. These are optimal processing speed in combination with various other system parameter settings that yield the greatest power savings. Somewhat surprising these results contrast to much conventional wisdom that underlies many protocols such as the “race-to-halt” mechanism. Race-to-halt suggests that one run the processor as fast as possible and then shut it down. In contrast, the sweet spots we identify show that it can be more power-efficient (for a given computational performance target) to run the processor more slowly for longer. To develop these results, in this paper we first study the interplay between fixed-threshold reactive power control mechanism and DVFS to identify the optimal operation settings. The optimal settings also appear in the threshold mechanism with job batching, i.e., batching certain amount of jobs before system wake-up. We then extend the concept to the multi-server case where we consider the relation between server farm size and processing speed.

The rest of the paper is organized as follows. In Section II we present the server model. In Section III we present the analysis for the single server case. The multi-server case is discussed in Section IV. We conclude in Section V.

II. SERVER MODEL

We model each server as a computation entity that processes jobs. Each server is equipped with a DVFS mechanism. DVFS is a conventional method widely used to trade off power consumption with processing speed by changing operating voltage and clock frequency. We assume the clock frequency can be scaled by a factor $f \in [0, 1]$ and the time it takes to process each job under DVFS is exponentially distributed with mean $1/\mu f$. For simplicity, herein we assume the processing time for all jobs is independent and identically distributed. Setting $f = 1$ yields maximum processing speed $1/\mu$ and setting $f = 0$ stops the server from processing jobs, i.e., the server is in the clock-gated mode.

The dynamic power consumption of a system supporting DVFS is proportional to $V^2 f$ where V is the supply voltage and f is the clock frequency scaling factor. The supply voltage is determined by frequency and can be reduced if the clock frequency is also reduced. This results in a cubic reduction in power consumption. Therefore we model the power consumed by the server as $P_0 f^3 + C$ where P_0 is the maximum power draw from the computing entity itself, e.g. CPU. The second term C is the average power drawn by peripherals such as DRAM, hard disk drivers (HDDs), network interface card (NIC), etc. This can be thought of as the “infrastructure” cost

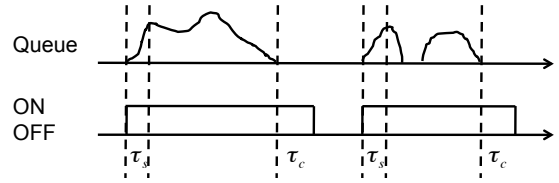


Fig. 2. Threshold mechanism.

incurred by keeping the computational unit on and ready to process jobs. Note that when $f = 0$, i.e., the server is in the clock-gated mode: the power consumed by the server is the peripheral power C . This is different from the mode that the server is shut-down in which case the power consumption is zero. When the server is shut-down, there is a wake-up penalty in terms of time and power. For the ease of illustration, we model the peripheral power C as independent of f . In practice, the peripheral power also depends on the system operation, exhibiting different values in active, idle and sleep modes [5].

III. SINGLE SERVER ANALYSIS

In this section we provide our analysis for the single server case. We assume jobs arrive according to a Poisson process with arrival rate λ . We study two conventional power-saving operations, namely the threshold mechanisms with and without job batching.

A. Threshold mechanism

We first describe the mechanism without job batching.

Definition 1 (Threshold Mechanism). *The server processes jobs until the queue is empty. Then it waits for a fixed amount of time threshold τ_c . If the next job arrives within this waiting threshold τ_c , the server processes the job and resumes normal operation. Otherwise the server shuts down in which the whole platform (CPU and the peripherals) is powered down consuming zero power. If the next job arrives after the waiting threshold τ_c (thus after the server has powered off), the server takes time τ_s to wake up before processing the job.*

A pair of sample paths illustrating the operation of this mechanism is provided in Figure 2. The upper sample path indicate queue occupancy. The lower sample path is binary, indicating when the server is on and off. For the ease of illustration, we assume whenever the server is not shut-down, its power consumption is consistent over time determined by the frequency scaling f . Our analysis can be easily extended to the case where the power spent in τ_c and τ_s are different from the normal operation.

Surprisingly for such a widely used mechanism, to the best of our knowledge, it has not been thoroughly studied from the queuing theoretic perspective. Indeed, it is not immediately clear how mean response time and power consumption are related under frequency scaling f and peripheral power C . In current implementation, the threshold value τ_c is chosen as a fixed value mostly based on operators’ own experience [10]. We investigate how the waiting threshold τ_c , frequency

scaling f and wake-up latency τ_s jointly affect the power and mean response time of such systems. We study this via a queueing theoretic analysis. Our results reveal that it is important to determine these operation parameters in a joint fashion. Naïvely picking τ_c too large or too small may lead to poor power efficiency.

The following theorem summarizes the relationship between mean response time $\mathbb{E}[R]$ and power consumption $\mathbb{E}[P]$.

Theorem 1. *The mean response time and mean power consumption of a server using the threshold mechanism are given by:*

$$\mathbb{E}[R] = \frac{1}{\mu f - \lambda} + \frac{2\tau_s + \lambda\tau_s^2}{2(e^{\lambda\tau_c} + \lambda\tau_s)} \quad (1)$$

$$\mathbb{E}[P] = (P_0 f^3 + C) \left(1 - \frac{1 - \frac{\lambda}{\mu f}}{e^{\lambda\tau_c} + \lambda\tau_s} \right). \quad (2)$$

Proof: It is shown that the mean response time for an M/G/1 queue with the first customer experiencing a random delay D is given by [15]:

$$\mathbb{E}[R] = \frac{1}{f\mu} + \frac{\lambda(1 + c_s^2)}{2f^2\mu^2 \left(1 - \frac{\lambda}{\mu f}\right)} + \frac{2\mathbb{E}[D] + \lambda\mathbb{E}[D^2]}{2(1 + \lambda\mathbb{E}[D])}, \quad (3)$$

where c_s^2 is the variance of coefficient. The random delay D in our case is $D = 0$ if $0 \leq T \leq \tau_c$ and $D = \tau_s$ if $T > \tau_c$ where T is the time elapse to see the first arrival after the server runs out of jobs. The random variable T is exponentially distributed with parameter λ . Therefore $\mathbb{E}[D]$ can be calculated as:

$$\mathbb{E}[D] = \int_{\tau_c}^{\infty} \tau_s \lambda e^{-\lambda t} dt = \tau_s e^{-\lambda\tau_c}. \quad (4)$$

Similarly, $\mathbb{E}[D^2] = \tau_s^2 e^{-\lambda\tau_c}$. Plugging them into (3) with $c_s^2 = 1$ for M/M/1 we obtain the mean response time (1).

The power expression can be derived as follows. Note that

$$\mathbb{E}[P] = (P_0 f^3 + C)(1 - f_{\text{off}}), \quad (5)$$

where f_{off} is the fraction of the time the server is off. Now consider a time duration L from the end of one epoch that the queue is empty to the end of next epoch that the queue is empty. Since this time duration starts with zero job and ends with zero job in the queue, the following equality holds:

$$\lambda L = \mu f \left(L - \frac{1}{\lambda} - \mathbb{E}[D] \right). \quad (6)$$

Within this time duration, the server will shut down only when next job arrives after τ_c . Thus f_{off} can be calculated as:

$$f_{\text{off}} = \frac{\int_{\tau_c}^{\infty} (t - \tau_c) \lambda e^{-\lambda t} dt}{L}. \quad (7)$$

Plugging in f_{off} into (5) we obtain the power consumption (2). ■

From Theorem 1, we have the following observations. First when $\tau_c = \infty$, the server never shuts down. The mean response time (1) and power consumption (2) reduce to:

$$\mathbb{E}[R] = \frac{1}{\mu f - \lambda} \quad \mathbb{E}[P] = (P_0 f^3 + C), \quad (8)$$

which is the mean response time and power consumption for an M/M/1 queue with frequency scaling f .

When $\tau_s = 0$, i.e., the server incurs no delay to wake up. The mean response time reduces to an M/M/1 case while the power consumption can be minimized by picking $\tau_c = 0$. Thus we have:

$$\mathbb{E}[R] = \frac{1}{\mu f - \lambda} \quad \mathbb{E}[P] = \frac{\lambda}{\mu f} (P_0 f^3 + C). \quad (9)$$

This means that if there is no cost to wake up a server, the server should shut down immediately when the queue becomes empty. However, note that the power-delay tradeoff is not monotonic: there is an optimal frequency that minimizes the power consumption (c.f. Figure 4). In other words, it is not always the case that running slow (while incurring large delay) leads to more power savings.

For a fixed nonzero τ_s , there is an optimal (τ_c, f) pair that minimizes the power consumption for a given delay performance. To see this, fix $\mathbb{E}[R] = R'$ and from (1) we obtain the relationship between f and τ_c :

$$\frac{1}{e^{\lambda\tau_c} + \lambda\tau_s} = \frac{2}{2\tau_s + \lambda\tau_s^2} \left(R' - \frac{1}{\mu f - \lambda} \right). \quad (10)$$

Plugging it into (2), we see that the optimal frequency scaling f is the one that minimizes the following:

$$\mathbb{E}[P] = (P_0 f^3 + C) \left[1 - \frac{2 \left(1 - \frac{\lambda}{\mu f} \right)}{2\tau_s + \lambda\tau_s^2} \left(R' - \frac{1}{\mu f - \lambda} \right) \right]. \quad (11)$$

Thus we have an optimal (τ_c, f) pair (c.f. Figure 5). This suggests that one should not set τ_c and f independently: they are coupled and depend on the quality of service requirement.

The race-to-halt mechanism is a special case of this threshold mechanism with $\tau_c = 0$ and $f = 1$. That is, the server runs as fast as it could when the queue starts to build up and shuts down immediately after it clears all the jobs. The mean response time and power consumption reduce to:

$$\mathbb{E}[R] = \frac{1}{\mu - \lambda} + \frac{\tau_s}{2(1 + \lambda\tau_s)} + \frac{\tau_s}{2} \quad (12)$$

$$\mathbb{E}[P] = (P_0 + C) \left(1 - \frac{1 - \frac{\lambda}{\mu}}{1 + \lambda\tau_s} \right). \quad (13)$$

The power consumption (13) is a monotonically increasing function with respect to λ . However for mean response time, there is a λ that minimizes the delay.

B. Threshold mechanism with job batching

In this section we extend our analysis to consider the threshold mechanism with job batching.

Definition 2 (Threshold Mechanism with Job Batching). *This mechanism is the same as the one in Definition 1 with the following difference. When the shut-down server sees the first job arrival, the server remains shut-down for some additional time τ_w before waking up. As before, wake-up takes times τ_s .*

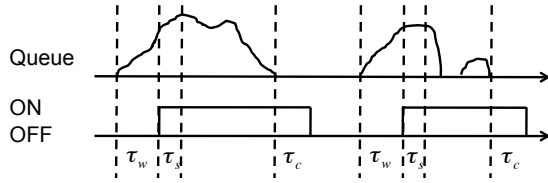


Fig. 3. Threshold mechanism with job batching.

As we did for the basic threshold mechanism, in Figure 3 we provide a pair of sample paths illustrating the operation of the modified mechanism. The upper sample path indicates queue occupancy. The lower sample path is binary, indicating when the server is on and off. Note the additional parameter vis-à-vis the basic mechanism. The intuition behind this mechanism is that by batching more jobs at the beginning, it is less likely that the server will run out of jobs in the near future. This mechanism is the spirit in the periodic power-on and power-off operation in MapReduce clusters and the idea of batching database queries (see [9] and the references therein). However, it is not clear how τ_w affects power and mean response time and the relation with f , τ_c and τ_s is unknown. We derive the mean response time and power consumption for this threshold mechanism with job batching in Lemma 2.

Lemma 2. *The mean response time and power consumption of the threshold mechanism with job batching are*

$$\mathbb{E}[R] = \frac{1}{\mu f - \lambda} + \frac{2(\tau_s + \tau_w) + \lambda(\tau_s + \tau_w)^2}{2(e^{\lambda\tau_c} + \lambda(\tau_s + \tau_w))} \quad (14)$$

$$\mathbb{E}[P] = (P_0 f^3 + C) \left(1 - \frac{(1 + \lambda\tau_w) \left(1 - \frac{\lambda}{\mu f}\right)}{e^{\lambda\tau_c} + \lambda(\tau_s + \tau_w)} \right). \quad (15)$$

Proof: The proof follows from the one in Theorem 1. In particular, the random delay D now becomes $D = 0$ if $0 \leq T \leq \tau_c$ and $D = \tau_s + \tau_w$ if $T > \tau_c$. We obtain (14) by solving for $\mathbb{E}[D]$ and $\mathbb{E}[D^2]$ and plugging in (3) with $c_s^2 = 1$. The power consumption can also be derived in the same way as in Theorem 1 with f_{off} replaced by:

$$f_{\text{off}} = \frac{\int_{\tau_c}^{\infty} (t - \tau_c) \lambda e^{-\lambda t} dt + \int_{\tau_c}^{\infty} \tau_w \lambda e^{-\lambda t} dt}{L}. \quad (16)$$

The rest of the proof follows from the one in Theorem 1. ■

Note that when $\tau_w = 0$, the system reduces to the threshold mechanism. When τ_w is very large, the system waits long period of time before waking up: the mean response time thus goes unbounded and the power consumption converges to $\frac{\lambda}{\mu f} (P_0 f^3 + C)$.

Under a certain mean response time budget $\mathbb{E}[R] = R'$, there is an optimal triple (τ_c, f, τ_w) that minimizes the power consumption. In particular, when $\tau_c = 0$, the mean response

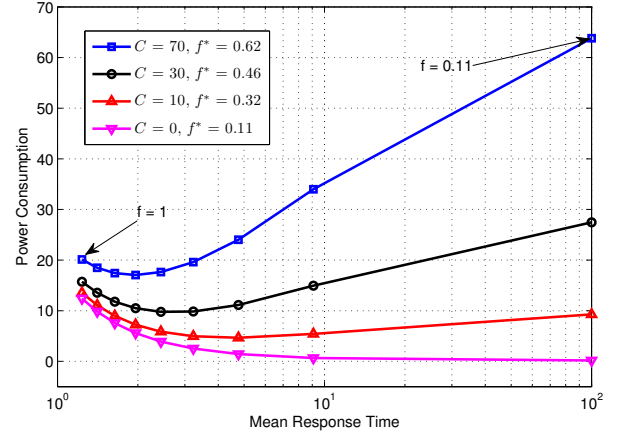


Fig. 4. Threshold mechanism, $\tau_s = \tau_c = 0$. Note that there is an optimal frequency f^* that minimizes the power consumption.

time and power consumption reduce to:

$$\mathbb{E}[R] = \frac{1}{\mu f - \lambda} + \frac{2(\tau_s + \tau_w) + \lambda(\tau_s + \tau_w)^2}{2(1 + \lambda(\tau_s + \tau_w))} \quad (17)$$

$$\mathbb{E}[P] = (P_0 f^3 + C) \left(1 - \frac{(1 + \lambda\tau_w) \left(1 - \frac{\lambda}{\mu f}\right)}{1 + \lambda(\tau_s + \tau_w)} \right). \quad (18)$$

Further with $f = 1$, the threshold mechanism reduces to the race-to-halt mechanism with job batching. We simulate its mean response time (17) and power consumption (18) in Figure 6.

C. Simulation Results

In this section we present our simulation results for the fixed threshold mechanisms. We choose the simulation parameters in real data traces from many literatures (see [5] and the references therein).

1) *Threshold mechanism:* We consider a computing facility with $P_0 = 150$, $\mu = 1$ and $\lambda = 0.1$ which models low utilization scenario. If the wake-up cost is negligible, i.e., $\tau_s = 0$, then from previous analysis we have $\tau_c = 0$ and the mean response time and power consumption reduce to (9). Figure 4 illustrates the power-delay tradeoff for various C when $\tau_s = \tau_c = 0$. Notice that there is an optimal frequency scaling that minimizes the power consumption. The results suggest that running jobs at large delay (using low frequency) may actually consume more power to run.

In a more realistic scenario where $\tau_s \neq 0$, Figure 5 validates our argument that there is an optimal (τ_c, f) pair that jointly minimizes the power consumption given a target mean response time (c.f. (11)). We set $P_0 = 150$, $C = 70$, $\lambda = 0.1$, $\mu = 1$ and $\tau_s = 10$. Notice that for a given mean response time R' , there is an optimal τ_c and an associated frequency scaling f that minimize the power consumption. Note also that for the mean response time achieved by the race-to-halt mechanism ($\tau_c = 0$, $f = 1$), we can pick another (τ_c, f) pair that yields smaller power consumption.

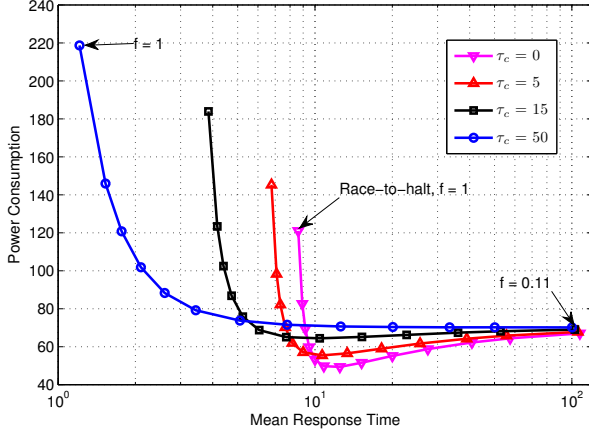


Fig. 5. Threshold mechanism, $\tau_s = 10$. Different target delay corresponds to different τ_c and f pair.

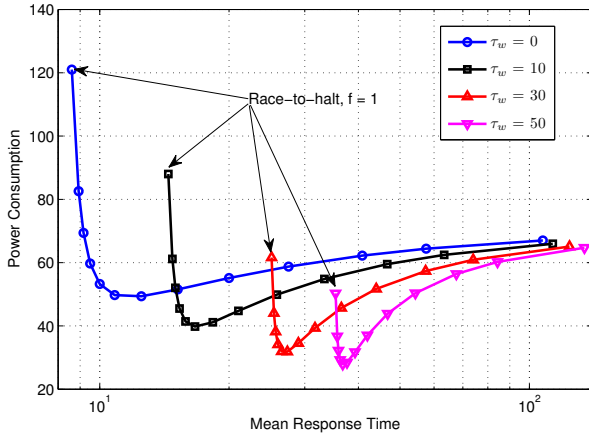


Fig. 6. Threshold mechanism with job batching, $\tau_s = 10$ and $\tau_c = 0$. Different target delay corresponds to different τ_w and f pair. Note that the curve with $\tau_w = 0$ is the same as the one with $\tau_c = 0$ in Figure 5.

2) *Threshold mechanism with job batching*: We simulate the mean response time (17) and power (18) for the threshold mechanism with job batching. We set $P_0 = 150$, $C = 70$, $\lambda = 0.1$, $\mu = 1$, $\tau_s = 10$ and $\tau_c = 0$. The frequency scaling f and batching period τ_w are kept as variables. The power-delay tradeoff is shown in Figure 6. Notice that for some mean response time achieved by the race-to-halt mechanism, we can pick another (τ_c, f) pair that yields smaller power consumption. The intuition is that to save power, one typically prefers smaller f over $f = 1$. However to maintain the same delay performance one needs to compensate the increase in delay caused by the smaller f by picking a smaller τ_w . Meanwhile, one should not decrease f too much either as doing so the peripheral power C will soon be the dominating factor. We also note that the power-delay tradeoff is monotonic for race-to-halt scheme: increasing τ_w always incurs larger delay and lower power consumption.

IV. MULTI-SERVER ANALYSIS

In this section we extend our queuing analysis to study the interplay between frequency scaling and facility plant size, i.e., the number of servers. We study two simple multi-server scenarios, namely flow splitting and job splitting. We observe that even in such simple settings there are optimal operating frequency and plant size pairs that minimize the power consumption.

Consider n parallel homogeneous servers with a centralized job dispatcher. Jobs arrive at the dispatcher according to a Poisson process with rate λ . The job dispatcher distributes jobs to servers according to some rules. In this section we consider two simple rules: flow splitting using Bernoulli splitting and job splitting using fork-join. We assume all servers use the same operating frequency scaling f , each consuming $P_0 f^3 + C$ amount of power.

A. Flow splitting

In the flow splitting case, the job dispatcher sends jobs to servers according to a Bernoulli splitting manner. Each server behaves as an M/M/1 queue with Poisson arrival rate λ/n .

Lemma 3. *The mean response time and power consumption of flow splitting multi-server system are:*

$$\mathbb{E}[R] = \frac{1}{f\mu - \frac{\lambda}{n}} \quad \mathbb{E}[P] = n(P_0 f^3 + C). \quad (19)$$

For any given $\mathbb{E}[R] = R'$, simple algebraic calculations show that there is an optimal frequency scaling and plant size pair that minimizes the power consumption. In particular, in large delay region $R' = \infty$, the optimal frequency scaling f and plant size n are given by:

$$f = \sqrt[3]{\frac{C}{2P_0}} \quad n = \frac{\lambda}{\mu f}. \quad (20)$$

This suggests that for power-efficient computation, it is not necessarily true that running as fast as possible or consolidating jobs onto as few servers as possible offers a better power efficiency. This phenomenon is visualized in Figure 7. We conjecture that similar observations exist for round robin scheduling where the inter-arrival time between jobs is Erlang- n distributed.

B. Job splitting

In the job splitting case, upon a job arrival the job dispatcher immediately makes n copies of the job and forks them in parallel to n servers. This models the queries to content retrieval databases where each incoming request can be simultaneously routed to n databases waiting for some of them to respond. Servers process requests in parallel and one queue is maintained at each server. When any k out of n servers respond, the rest of $n - k$ servers abandon the corresponding requests and the job departs the system. Such system is often termed (n, k) fork-join queue [16] in queuing theory literature. There is no known close form solution for the mean response time of the fork-join system, not even for (n, n) system. However several bounds exist (for example, see

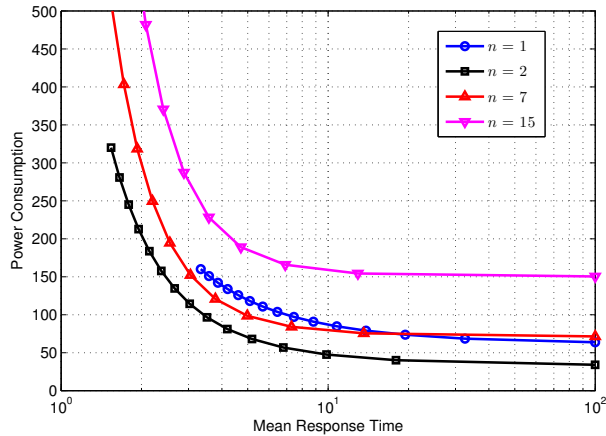


Fig. 7. Power delay tradeoff with flow splitting.

[17]). For the job splitting case, working with the bounds we notice that there is also an optimal frequency scaling f and plant size n combination such that the power is minimized for a given delay budget.

In both flow splitting and job splitting cases, packing jobs onto fewer servers requires faster processor speed to maintain a given delay performance thus increasing the processing power $P_0 f^3$. On the other hand, provisioning more servers always incurs the fixed peripheral power expenditure C .

C. Simulation Results

We simulate the mean response time and power consumption for multi-server flow splitting case. The case for job splitting shares the same spirit (omitted due to page limits). We set $P_0 = 150$, $C = 10$, $\lambda = 0.7$ and $\mu = 1$ while the frequency scaling f and the number of servers n are kept as variables. Simulation results are shown in Figure 7. For each n , we simulate different frequency scaling f to plot the curve. Note that for some fixed mean response time, the power consumption first decreases then increases with increasing n . Intuitively, in one extreme case where jobs can tolerate large delay, the system should run slowly with small amount of servers (c.f. (20)). In another extreme case where jobs demand fast response, the system should run faster with many servers powered on.

V. CONCLUSION AND FUTURE WORK

In this paper we present a queuing theoretic analysis of some widely used power-efficient operations in modern computing. We analytically characterize the power-delay tradeoff for the threshold mechanisms with and without job batching. We also analyze the multi-server case. For these mechanisms we discover that there oftentimes exist sweet spots: optimal combinations of processing speed and other system parameters that yield best power efficiency.

There are many promising future directions. These include the investigation of other power-efficient mechanisms. For the

single server case, we will consider predictive wake-up and shut-down routines (c.f. Figure 1). Such proactive control requires some prediction tools to predict traffic and offers improvements in delay. For the multi-server case, we question the power efficiency of many conventional dispatching algorithms as most of them are not traditionally designed for power-efficient computing. We would like to understand the interplay and investigate the optimality between dispatching mechanisms and other system parameters. This will motivate some design guidances for power-efficient job dispatching routines.

REFERENCES

- [1] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *ACM SIGCOMM Computer Communication Review*, vol. 39, pp. 68–73, Dec. 2008.
- [2] V. Mathew, R. K. Sitaraman, and P. Shenoy, "Energy-aware load balancing in content delivery networks," *IEEE International Conference on Computer Communications (INFOCOM)*, pp. 954–962, Mar. 2012.
- [3] US Environmental Protection Agency – Energy Star Program, "Report to Congress on server and data center energy efficiency public law 109-431," Aug. 2007.
- [4] A. Verma, P. Ahuja, and A. Neogi, "pMapper: power and migration cost aware application placement in virtualized systems," *Proceedings of the ACM/IFIP/USENIX International Conference on Middleware*, pp. 243–264, Dec. 2008.
- [5] D. Meisner, B. T. Gold, and T. F. Wenisch, "PowerNap: eliminating server idle power," *ACM Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 205–216, Mar. 2009.
- [6] P. Bodik, M. P. Armbrust, K. Canini, A. Fox, M. Jordan, and D. A. Patterson, "A case for adaptive datacenters to conserve energy and improve reliability," Tech. Rep. UCB/ECS-2008-127, Sep. 2008.
- [7] A. Beloglazov, R. Buyya, Y. C. Lee, and A. Zomaya, "A taxonomy and survey of energy-efficient data centers and cloud computing systems," Tech. Rep. arXiv:1007.0066, Sep. 2010.
- [8] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch, "Power management of online data-intensive services," *ACM Proceedings of International Symposium on Computer Architecture (ISCA)*, pp. 319–330, Jun. 2011.
- [9] W. Lang and J. M. Patel, "Energy management for MapReduce clusters," *Proceedings of the VLDB Endowment*, vol. 3, pp. 129–139, Sep. 2012.
- [10] N. Madan, A. Buyuktosunoglu, P. Bose, and M. Annavaram, "A case for guarded power gating for multi-core processors," *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 291–300, Feb. 2011.
- [11] A. Gandhi and M. Harchol-Balter, "How data center size impacts the effectiveness of dynamic power management," *Allerton Conference on Communication Control and Computing*, pp. 1164–1169, Sep. 2011.
- [12] M. J. Neely, "Lower power dynamic scheduling for computation systems," Tech. Rep. arXiv:1112.2797, Dec. 2011.
- [13] Y. Yao, L. Huang, A. Sharma, L. Golubchik, and M. J. Neely, "Data centers power reduction: a two time scale approach for delay tolerant workloads," *IEEE International Conference on Computer Communications (INFOCOM)*, pp. 1431–1439, Mar. 2012.
- [14] M. J. Neely, A. S. Tehrani, and A. G. Dimakis, "Efficient algorithms for renewable energy allocation to delay tolerant consumers," *IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pp. 549–554, Oct. 2010.
- [15] P. D. Welch, "On the generalized M/G/1 queueing process which the first customer of each busy period receives exceptional service," *Operation Research*, vol. 12, pp. 736–752, Sep. 1964.
- [16] R. Nelson and A. Tantawi, "Approximate analysis of fork/join synchronization in parallel queues," *IEEE Transactions on Computers*, vol. 37, pp. 739–743, Jun. 1988.
- [17] G. Joshi, Y. Liu, and E. Soljanin, "Coding for fast content download," *Allerton Conference on Communication, Control and Computing*, Sep. 2012.